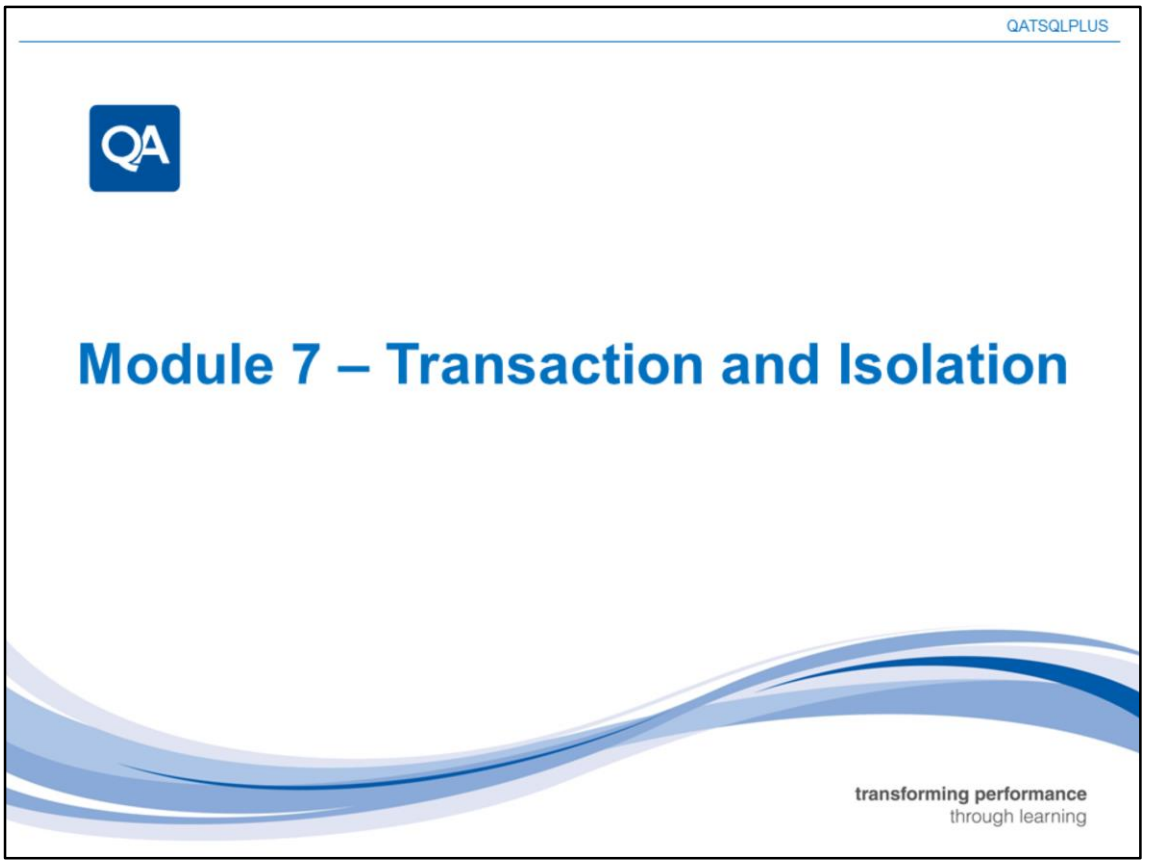
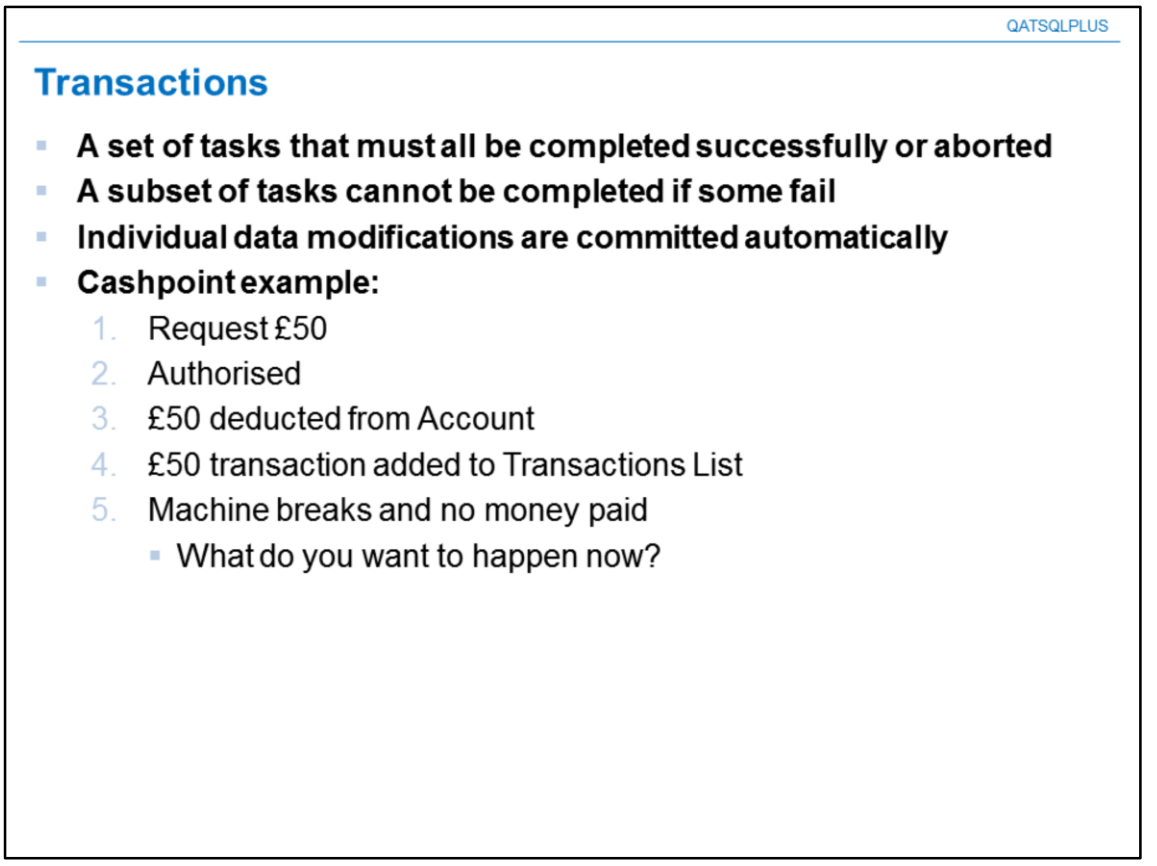

QATSQLPLUS
Module 7 – Transaction and Isolation
transforming performance
through learning

QATSQLPLUS

Transaction and Isolation Levels

- **Transactions**
  - Overview
  - Implementing transactions
  - Session settings
- **Isolation levels**
  - Overview
  - Changing the isolation level
  - Table hints



Transactions

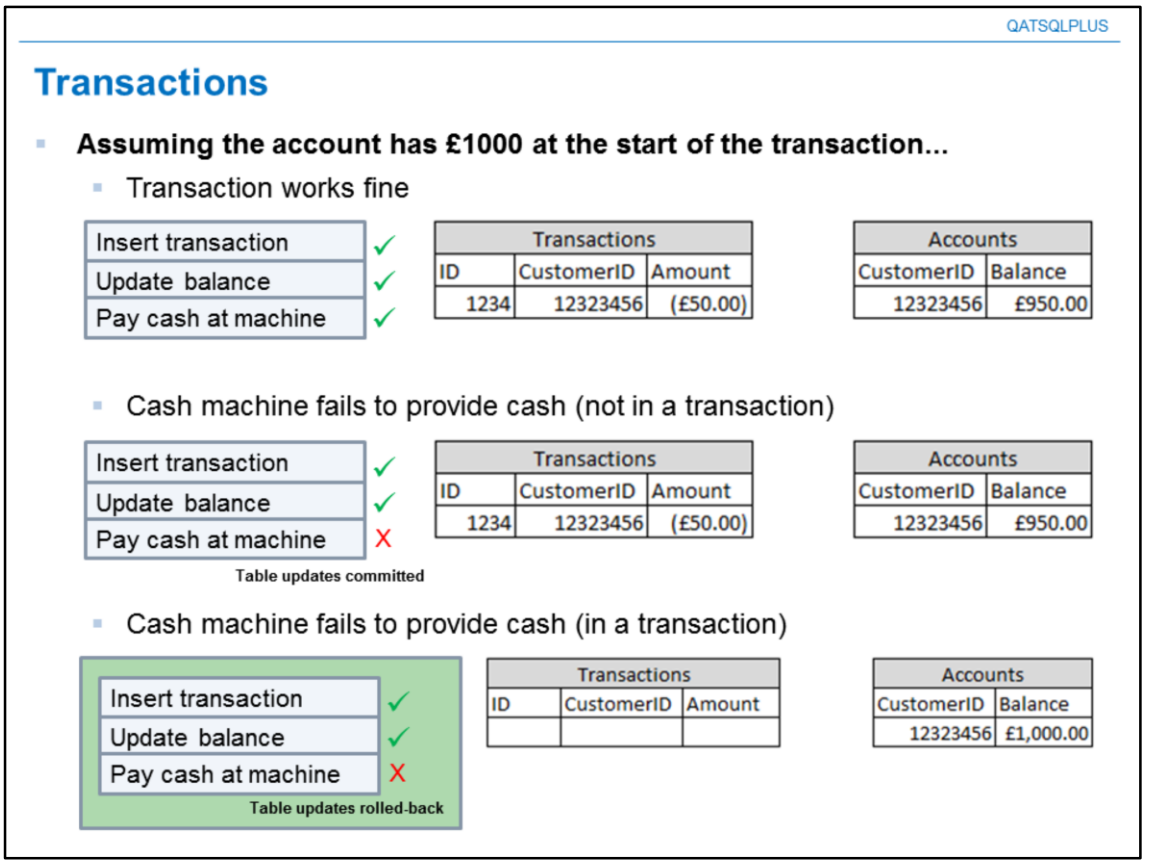
- A set of tasks that must all be completed successfully or aborted
    - A transaction can be committed (save all changes) or rolled-back (undo all changes)
- A subset of tasks cannot be completed if some fail
    - Partial completion of a transaction is not allowed
- Individual data modifications are committed automatically
    - As the individual statements are not covered by a transaction, each is treated as a transaction, so does not affect any other statements

Examples

1. Exchanging group of 3 seats in a stadium
    - Allocate customers to seats B1,B2,B3
    - Remove the customers from seats A50,A51,A52
    - Possible outcomes without a transaction covering both steps
        1) Customer has 6 seats if the remove step fails
        2) Customer has no seats if the allocate step fails

2. Money transfer
    - Amount is deducted from Account A
    - Amount is added to Account B

    - Possible outcomes without a transaction covering both steps
        1) Customer has extra money if the deduct step fails
        2) Customer has less money if the add step fails



In the transaction above the three steps are:

1. Insert a row into the banking transactions table showing amount requested.
2. Update the accounts row for the customer deducting the amount.
3. Pay the money at the cash machine.

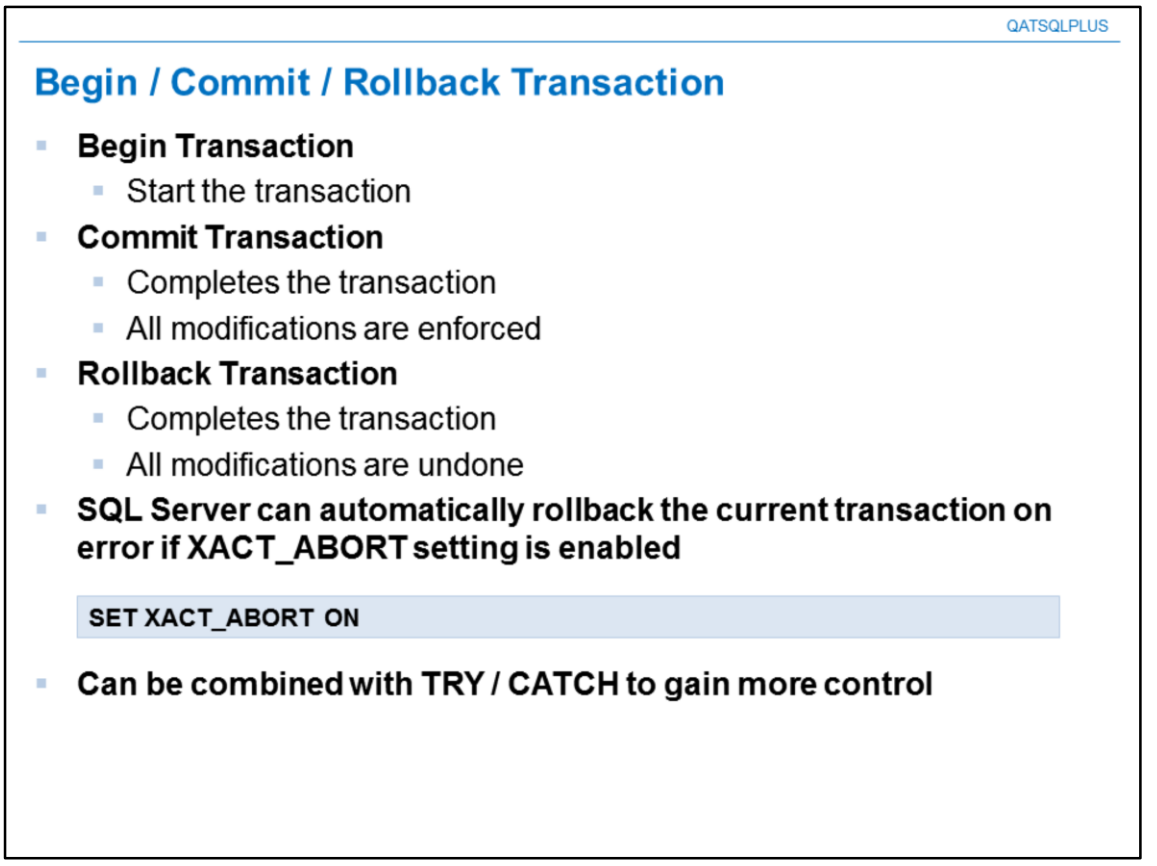
Each step should be in a transaction so that if any step fails, none of the steps will be enacted. If a step fails, the transaction should fail immediately and not process subsequent steps.

Possible outcomes

1) Insert transaction succeeds, balance update succeeds, pay money step succeeds
    - Outside a transaction = transaction and account records are consistent, and customer has been delivered money
    - Inside a transaction = transaction and account records are consistent, and customer has been delivered money
2) Insert transaction fails, other two steps do not execute
    - Outside a transaction = transaction and account records are inconsistent, and no money delivered
    - Inside a transaction = transaction is rolled back, so records are consistent

3) Insert transaction succeeds, balance update fails, other step does not execute
    - Outside a transaction = transaction and account records are inconsistent, but no money delivered
    - Inside a transaction = records are rolled-back entirely
4) Insert transaction succeeds, balance update succeeds, pay money fails
    - Outside a transaction = transaction and account records are consistent, but no money delivered, so customer down £50
    - Inside a transaction = records are rolled-back entirely, so customer not down any money

As shown in the possible outcomes, a transaction protects the database and other tasks against tasks failing. If all steps succeed, then there is no difference between having or not having a transaction.



Each explicit transaction should have two or three transaction related statements.

- **Begin Transaction** – marks the start of the transaction
    - Any data modification statement after this point would be held as a part of the transaction and liable to be rolled-back if required
- **Commit Transaction** – completes the transaction and all modifications are enforced
    - Once a transaction has been committed it cannot be rolled-back
- **Rollback Transaction** – completes the transaction but all modifications are undone
    - Once a transaction is rolled-back it cannot be committed

SQL Server can automatically rollback the current transaction on error if XACT_ABORT setting is enabled. Any statement after the begin transaction that fails at runtime will cause the transaction to be automatically rolled-back.

The developer can combine transactions within a TRY / CATCH block to gain more control. If any error is raised within the TRY block then processing will switch to the CATCH block. Usually the COMMIT TRAN statement is placed within the TRY block and the ROLLBACK TRAN statement is placed within the CATCH block.

Transaction Example

Command outline:

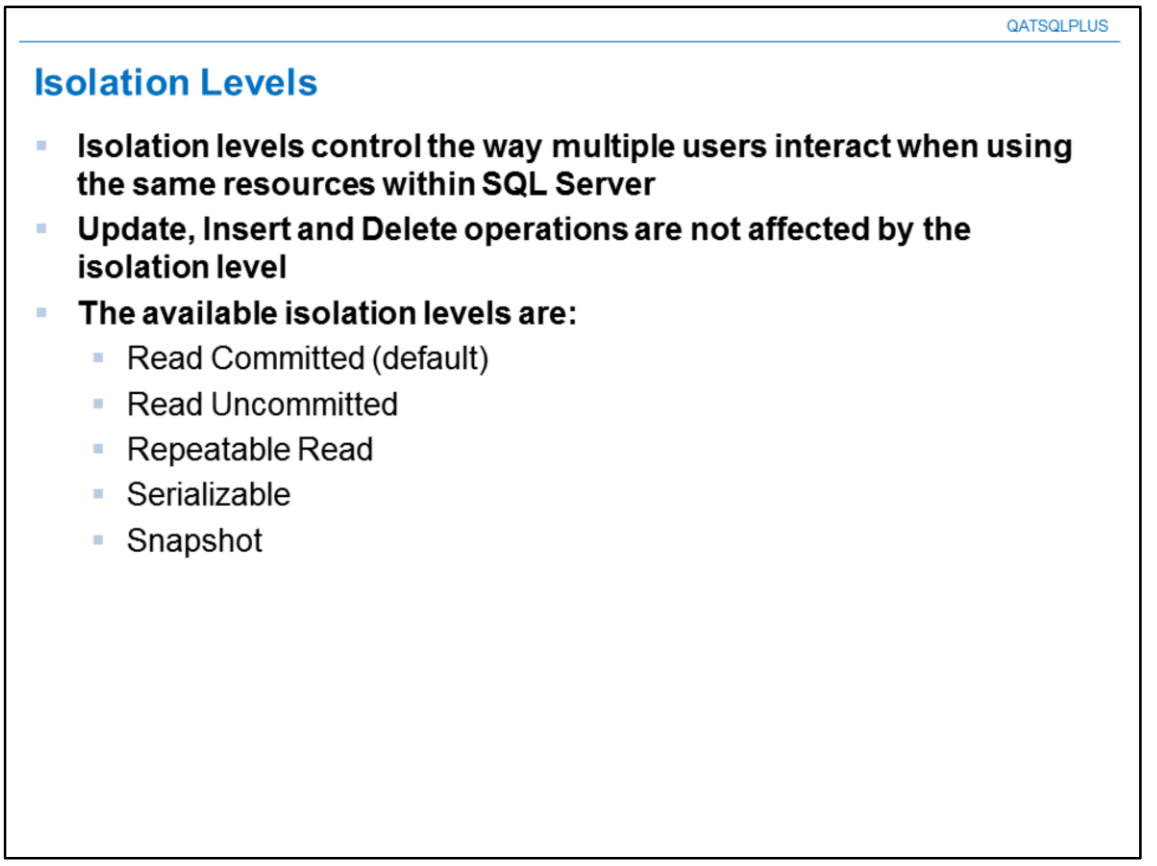
```
BEGIN TRY
    BEGIN TRAN
        <some data modifications>
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN
    <code for the error>
END CATCH
```

Transaction Example (cont.)

Demonstration:

```
DECLARE @AccountNo INT = 12343422
DECLARE @Machine INT = 23321
DECLARE @Amount MONEY = 50.00

BEGIN TRY
    BEGIN TRAN
        INSERT INTO dbo.CashpointTrans(AcctNo, TranDT, MachineID, TranValue)
            VALUES (@AccountNo,getdate(),@Machine, @Amount)
        UPDATE dbo.CurrentAccount
            SET Balance = Balance – @Amount
            WHERE AcctNo = @AccountNo
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN
    THROW
END CATCH
```



Isolation Levels

- Isolation levels control the way multiple users interact when using the same resources within SQL Server
- Each statement that accesses data usually needs to be granted a lock before reading or updating data tables
- With a SELECT statement the lock will be a shared lock by default
    - Exclusive locks and shared locks are not compatible, so an object cannot have both types granted simultaneously
- Update, Insert and Delete operations are minimally affected by the isolation level as each data modification step takes exclusive locks on the objects or rows to be modified
- The available isolation levels are:
    - Read Committed (default)
    - Read Uncommitted
    - Repeatable Read
    - Serializable
    - Snapshot

Each of the isolation levels will be covered in the following pages.

Read Committed

- **Read Committed will only allow rows to be selected that are not currently exclusively locked**
- **The read operation will be blocked until the lock is released**
- **Advantages:**
  - Rows are read in a consistent state
- **Disadvantages:**
  - May slow select operations
  - Does not hold locks within a transaction, so can result in inconsistent querying if multiple queries access the same rows

Read Uncommitted

- **Read Uncommitted will allow rows to be selected regardless of their lock state**
- **The read operation will not be blocked**
- **Advantages:**
  - Quick return of select results
- **Disadvantages:**
  - Rows being read in an inconsistent state
  - Rows being read that are never committed to the table

QATSQLPLUS

Repeatable Read

- **Repeatable Read will only allow rows to be selected that are not currently exclusively locked**
- **The read operation will be blocked until the lock is released**
- **All rows selected will be locked to prevent any future changes within the current transaction**
- **Advantages:**
  - Rows are read in a consistent state
  - Holds locks within a transaction, so previously read rows will still be consistent if the query is repeated in the same transaction
- **Disadvantages:**
  - May slow concurrent operations
  - Does not guarantee that more rows will be returned in subsequent queries

Serializable

- **Serializable will only allow rows to be selected that are not currently exclusively locked**
- **The read operation will be blocked until the lock is released**
- **All rows selected will be locked to prevent any future changes within the current transaction, and no rows will be allowed to be added that would have been returned in the previous queries**
- **Advantages:**
  - Rows are read in a consistent state
  - Holds locks within a transaction, so same result set will be returned consistently
- **Disadvantages:**
  - May slow concurrent operations

Snapshot

- **Snapshot will allow all current rows to be selected in the state at the start of the transaction**
  - New inserted rows will not be returned
  - Updated and deleted rows will be shown as at the start of the transaction
- **The read operation will be not be blocked**
- **Advantages:**
  - Rows are read in a consistent state
  - Fast reading of data
- **Disadvantages:**
  - Needs to be enabled on the database first
  - Overhead on the server keeping the old rows

Exercise